MARYLIE/IMPACT

A Parallel Beam Dynamics Code with Space Charge based on the MaryLie Lie Algebraic Beam Transport Code and the IMPACT Parallel Particle-In-Cell Code *

Robert D. Ryne
Ji Qiang

Accelerator and Fusion Research Division
Lawrence Berkeley National Laboratory
Berkeley, California 94720

Alex J. Dragt

Department of Physics and Astronomy
University of Maryland
College Park, MD 20742

Salman Habib

Theoretical Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Filippo Neri
C. Thomas Mottershead
Peter Walstrom

LANSCE Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Viktor Decyk

Physics Deparment
Univ. of California, Los Angeles
Los Angeles, CA 12345

Roman Samulyak

Center for Data Intensive Computing
Brookhaven National Laboratory
Upton, New York 11973

October, 2003

*Work supported in part by U.S. Department of Energy, Office of Science, Office of High Energy Physics and Office of Advanced Scientific Computing Research under the auspices of the Scientific Discovery through Advanced Computing (SciDAC) program and by grant DEFG02-96ER40949.

Contents

1 Introduction to MaryLie/IMPACT **2**
1.1 Brief Program Description 2
1.2 Brief Description of the ML/I Front End 3
1.3 Specification of Units in ML/I 4
1.4 Summary of New Features 5
1.5 Program Availability 6

2 Catalog of New MaryLie/IMPACT commands **7**
2.1 Automatic slicing of thick elements 9
2.2 Automatic application of a command or string of commands 10
2.3 Automatic tracking 11
2.4 Automatic concatenation 12
2.5 Set beam parameters 13
2.6 Print particle coordinates and momenta 14
2.7 Set parameters of Poisson solver for space charge calculation 15
2.8 Print 1D Beam Profile 16
2.9 rayscale: scale particle data 17
2.10 raytrace: trace rays 18
2.11 resetarclength: reset arc length 19
2.12 symbolparser: control treatment of undefined symbols in input file 20

3 MaryLie/IMPACT Examples **21**
3.1 Overview 21

Chapter 1

Introduction to MaryLie/IMPACT

1.1 Brief Program Description

MaryLie/IMPACT (ML/I) is a hybrid code that combines the beam optics capabilities of MaryLie with the parallel Particle-In-Cell (PIC) capabilities of IMPACT. As such, it can be used to model beam dynamics in a wide range or rf accelerators, including linear and circular accelerators, with or without space charge, including acceleration.

MLI/I was developed by taking certain routines (e.g. space-charge routines and rf cavity routines) out of IMPACT and incorporating them in MaryLie. MaryLie itself was modified to automatically slice elements and, using split-operator methods, MaryLie was then given the ability to perform space-charge kicks in between slices. This enables ML/I to track particles using a split-operator integration algorithm under the assumption that the single-particle Hamiltonian is given by $H = H_{ext} + H_{sc}$, where H_{ext} is the portion of the Hamiltonian corresponding to the external fields and where H_{sc} is the portion corresponding to the space-charge fields (in the mean-field approximation).

The use of ML/I is backward compatable with MaryLie. For example, all the MaryLie beamline elements, commands, map analysis capabilities, and fitting/optimizing capabilities, are present in ML/I. However, the ML/I front end has the ability to read lattice descriptions in both the MaryLie style and the Standard Input Format (SIF). This is an important new capability given the very large number of files written in the SIF style. In some cases the input format involves an augmentation of SIF, since certain MLI capabilities – notably the ability to treat space charge and the ability to automatically perform certain commands – requires additional parameters to some definitions. For example, every thick element can have an additional parameter denoting a number of slices that the element is to be cut into.

Users unfamiliar with MaryLie can download the MaryLie manual from the web pages of Alex Dragt's Dynamical Systems and Accelerator Theory group at the University of Maryland, http://physics.umd.edu/dsat . Users unfamiliar with the Standard Input Format can download documentation from the Methodical Accelerator Design (MAD) web pages at CERN, http://www.cern.ch/mad . This ML/I manual should be thought of as an addendum to be used in concert with the before-mentioned manuals.

1.2 Brief Description of the ML/I Front End

As already mentioned, the ML/I front end can read MaryLie-style input and it can also read beamline descriptions in the Standard Input Format. The latter is the format used in the MAD code from CERN, and has become a widely adopted standard. There are slight differences in the standard between MAD version 8 and MAD version 9; ML/I attempts to parse them both correctly without user intervention.

As stated previously, ML/I input is backward compatible with MaryLie. Therefore, all the examples in the MaryLie manual still run under ML/I. In fact, ML/I files can mix both MaryLie-style input and SIF-style input. A short description of MaryLie input and SIF input follows.

In MaryLie, beamline elements are defined in a menu (under #menu in the input file) on 2 successive lines of code:

```
userlabel typecode
value1 value2 ... valueN
```

For example, in MaryLie a quadrupole magnet could be defined using the following:

```
myquad quad
0.5 2.82 1 1
```

This defines a quadrupole named myquad with a length of 0.5m, a gradient of 2.82 T/m, and with leading and trailing fringe fields turned on. Note that all the parameters must be present (i.e. there are no defaults), and they have to occur in the order specified in the MaryLie manual. In the SIF format, beamline elements are defined in the following form,

```
userlabel: typecode, parameter1=value1, parameter2=value2,... parameterN=valueN
```

For example, a quadrupole could be defined using

```
myquad: quadrupole, l=0.5 k1=6.22
```

This describes a quadrupole named myquad having a length of 0.5 meter and a "strength" of 6.22 m^{-2}. Note that, in the Standard Input Format, parameters can occur in any order on the input line, and default values apply when no value is provided. This example also illustrates the fact that different conventions are sometimes used to specify quantities, e.g. the gradient on axis, or the focusing strength in m^{-2}. ML/I attempts to deal with such situations by allowing either quantity to be specified. For example, the symbol `g1` can be used to specify the gradient in Tesla/meter:

```
myquad: quadrupole, l=0.5 g1=12.3
```

Other valid examples (assuming brho has been defined) are:

```
myquad: quadrupole, l=0.5 g1=6.22*brho
myquad: quadrupole, l=0.5 k1=12.3/brho
```

The preceding dealt with the description of beamline elements. Lines (which are collections of elements) are treated as follows:

In MaryLie, lines are defined under the #lines section of the input file as follows,

```
userlabel
item1 item2 ... itemN
```

In the Standard Input Format the description is,

```
userlabel, line=(item1 item2 ... itemN)
```

At present, ML/I cannot parse input with nested parentheses, userlabel, line=(item1 item2 (item3 item4) item5)

In regard to the overall organization of the input file, ML/I omits some of the MaryLie requirements that the file be separated into sections of elements, lines, etc. Instead, elements, lines, and defined constants (which were not available in MaryLie) can be freely intermixed.

1.3 Specification of Units in ML/I

The issue of units is one that has no doubt caused many accelerator designers to have hours of frustration and lost productivity. ML/I attempts to address this by providing a flexible and systematic treatment of units.

Essentially all particle-based beam dynamics code deal with the six-vectors of coordinates and momenta (or coordinates and velocities for codes that do not use canonical variables). Let (x, p_x, y, p_y, t, p_t) denote such a six-vector in physical units. In other words, x and y have dimensions of length, t (deviation in arrival time) has the dimensions of time, p_x and p_y have dimensions of momenta, and p_t has dimensions of energy. Most beam dynamics codes (ML/I, MaryLie, MAD8, MAD9, etc.) use dimensionless variables. We will define these in terms of three scaling constants, l,δ, and ω. The dimensionless variables are $(x/l, p_x/\delta, y/l, p_y/\delta, \omega t, p_t/\omega l\delta)$. Note that, looking at each the products xp_x, yp_y, and tp_t, the scale factor relating the new product to the old product is the same in all three cases, i.e. the scale factor is $l\delta$. As a result the transformation between the new variables and the old variables is canonical.

The choice of the three scaling quantities l,δ, and ω determines the choice of dimensionless variables (i.e. the choice of units). For example, in MaryLie, l is chosen by the user, δ is defined to be p_0 (the reference momentum), and ω is automatically selected so that $\omega l/c = 1$. This is the natural choice for beam transport systems where there is no acceleration; we call these "magnetostatic" or "static" units. When acceleration is present, a different (constant) scale factor for the momentum is needed, since p_0 is changing. An obvious choice is to choose $\delta = mc$; we call this choice "dynamic" units. Note that IMPACT uses dynamic units with the added restriction that $\omega l/c = 1$ (where ω is specified by the user).

ML/I allows all three quantities l,δ, and ω to be selected by the user. This is accomplished via the *units* command. Valid examples are,

```
myunits: units, type=static
myunits: units, type=dynamic
myunits: units, l=5.0, p=2.5, w=3.6
```

The default scale length for ML/I is $l = 1$ m; the default scale angular frequency is $\omega = c/l = 299792458$ rad/sec; and the default units for ML/I are magnetic units. These values are the same as for MaryLie except that in MaryLie the user must specify the scale length.

Note that, when using the `units` command, the input file should not contain the MaryLie style #beam input (which contains both the beam parameters and the scale length); instead, the `beam` command (in the MAD style) should be used along with the `units` command.

1.4 Summary of New Features

As already mentioned, the following three items represent key features in the ML/I code:

- space charge: ability to model beams with space charge
- acceleration: ability to model accelerating beams
- SIF compatibility: ability to read lattices in the Standard Input Format

In addition, the synthesis of a beam optics code (MaryLie) with a particle-in-cell code (IMPACT) is enhanced by the addition of certain functionality that is useful for tracking and other purposes. This functionality is embodied in the following "automatic" commands:

- Automatic slicing: thick elements now have an additional parameter, `slices=`, for which the default value is one. This can have a number of uses. One important use is related to the inclusion of space charge. Namely, there can be a space-charge kick in the middle of every slice. Other uses involve autoslicing combined with automatic application of commands. This could be used, for example, to print lattice functions at points within thick elements.
- Automatic application of a command or string of commands: Using the `autoapply` command, the user can specify a menu element or the name of a line (sequence of commands) that is to be automatically applied before and after every slice. For example, if a user wanted to tabulate the rms moments as a function of distance along the beamline, a user could slice the elements and then preapply or postapply the commands to compute and print the moments.

Two more "automatic" commands available to the user are:

- Automatic tracking: Normally, ML/I concatenates maps whenever a beamline element is encountered. However, this behavior can be changed by using the `autotrack` command, which causes ML/I to track particles, with or without space charge, whenever an element is encountered.
- Automatic concatenation: The `autoconcat` command causes ML/I to switch from `autotrack` mode to the default mode of operation where beamline elements are automatically concatenated.

Finally, there are certain ML/I commands that are similar to MaryLie commands but provide added functionality. For example, ML/I has a `raytrace` command that is similar to the MaryLie `rt` command, but in addition provides user control of file names, control of the subset of particles to be printed, etc. In general, command names in ML/I are longer than names (usually 4 characters or less) used in MaryLie. Note that, because the normal MaryLie input style requires a specific number of parameters for each menu element (i.e. there are no defaults), it is difficult to enhance existing MaryLie commands (like `rt`) while maintaining backward compatibility with MaryLie. *

1.5 Program Availability

ML/I has been installed on NERSC's IBM/SP computer, seaborg.nersc.gov. The executable version is at `~ryne/MLI/mli.x`, and example files can be found in `~ryne/MLI/Examples`. Users who want full access to source and executables need to obtain permission to use both MaryLie (from Alex Dragt, dragt@physics.umd.edu) and MaryLie/IMPACT (from Robert Ryne, RDRyne@lbl.gov). Full access is available to DOE labs and to institutions performing research under DOE contracts. Access by other institutions is determined on a case-by-case basis.

*The one exception to this is that it is possible to use MaryLie style input and slice thick elements. This is accomplished by placing the `autoslice` command early in the master input file. Encountering this causes menu definitions in the MaryLie style to be parsed assuming that there is an additional (last) parameter that is equal to the number of slices.

Chapter 2

Catalog of New MaryLie/IMPACT commands

The computer code MaryLie contains a comprehensive collection of beamline elements and commands, and all of these are available, unchanged, in ML/I. In addition, ML/I contains many new commands and elements. Users will furthermore notice that some ML/I commands are similar to, but not identical to, certain MaryLie commands. For example, the "rt" command in MaryLie and the "raytrace" command in ML/I both deal with tracing rays. The reason for this and other partial duplications is that the ML/I commands add additional functionality. For example, the "raytrace" command adds the ability to write results to named files, to write results on a sequence of files, and to print only a subset of the particles in the output files. Rather than alter the original MaryLie commands and break backward compatibility with the large body of existing MaryLie files and examples in the MaryLie manual, the MaryLie commands have not been altered.

Below is a summary of the new ML/I commands: Parameters accepted for various elements and their default values (underlined) are shown in Table 1.

The new commands and their type code mnemonics are listed below. Also listed are the subsections that describe them in detail.

AUTOSLICE	SLICES=*# of slices*, L=*distance between slices*, CONTROL=`local`, `global`, *or* `none`
AUTOAPPLY	NAME=*name of menu element or line*
AUTOTRACK	TYPE=`taylor`*N* *or* `symplectic`*N*
AUTOCONCAT	
BEAM	PARTICLE=`proton`,`electron`, `positron`,*or* `H-`, MASS=*particle mass in GeV/c*^{2}, CHARGE=*particle charge in units of e*, ENERGY=*total energy in GeV*, EKINETIC=*kinetic energy in GeV*, PC=*total momentum in GeV/c*, BRHO=*magnetic rigidity in Tesla/meter*, GAMMA1=γ*-1*, GAMMA=γ, BCURR=*beam current in Amperes*
,	BFREQ=*frequency of bunches in Hz*, MAXRAY=*maximum number of particles*
PARTICLEDUMP	MIN=, MAX=, SEQUENCELENGTH=, PRECISION=, FILE=, CLOSE=`true` *or* `false`, FLUSH=`true` *or* `false`
POISSON	NX=, NY=, NZ=, NGRIDPOINTS=`fixed` *or* `variable`, BOUNDINGBOX=`fixed` *or* `variable`, XMIN=, XMAX=, YMIN=, YMAX=, ZMIN=, ZMAX=, XBOUNDARY=`dirichlet`, `periodic`, *or* `open`, YBOUNDARY=`dirichlet`, `periodic`, *or* `open`, ZBOUNDARY=`dirichlet`, `periodic`, *or* `open`
PROFILE1D	COLUMN=, BINS=, SEQUENCELENGTH=, PRECISION=, RWALL=, FILE=, CLOSE=`true` *or* `false`, FLUSH=`true` *or* `false`
RAYSCALE	XMULT=, PXMULT=, YMULT=, PYMULT=, TMULT=, PTMULT=, XDIV=, PXDIV=, YDIV=, PYDIV=, TDIV=, PTDIV=
RAYTRACE	NORDER=, NTRACE=, NWRITE=, SEQUENCELENGTH=, PRECISION=, MIN=, MAX=, FILE1=,FILE2=, CLOSE=`true` *or* `false`, FLUSH=`true` *or* `false`
RESETARCLENGTH	
SYMBOLPARSER	DEFAULTZERO=`true` *or* `false`
UNITS	TYPE=`static` *or* `dynamic`, L=*scale length*, P=*scale momentum in GeV/c*, F=*scale frequency in Hz*, T=*scale time in sec*, W=*scale angular frequency in rad/sec*

Table 2.1: MLI command parameters and default values

Type Code	Element	Subsection
autoslice	Automatic slicing of thick elements	2.1
autoapply	Automatically apply commands at the end of each slice	2.2
autotrack	Automatically track particles at the end of each slice	2.3
autoconcat	Automatically concatenate maps	2.4
beam	Set parameters for a reference particle and for macroparticles defined with respect to that reference particle	2.5
particledump	Print particle data to a file	2.6
poisson	Set parameters of Poisson solver	2.7
profile1d	Print 1D profile of a phase space variable	2.8
rayscale	Scale particle data	2.9
raytrace	Perform a ray trace	2.10
resetarclength	reset the arc length to zero	2.11
symbolparser	Governs treatment of undefined symbols in input file	2.12
units	Specify units (i.e. scale length, scale momentum, and scale time	2.13

2.1 Automatic slicing of thick elements

Type Code: autoslice

Parameters:

1. *slices:* number of slices
2. *l:* distance between slices (m) (if *slices* are not specified in the parameter list)
3. *control:* = `none` to turn off autoslicing; = `local` to specify slices on a per-element basis (default); = `global` to specify a global number of slices or globally defined distance between slices.

Examples:

```
myslice1: autoslice, control=local
myslice2: autoslice, control=global, slices=2
myslice3: autoslice, control=global, l=0.01
```

The first example above specifies that autoslicing of elements will occur, and that the number of slices is to be specified separately for each element when the elements are defined in the input file. (The default number of slices for thick elements is one).

The second example above specifies that autoslicing of elements will occur, and that every thick element will be cut into two slices. Note that, even if the number of slices has been specified for the thick beamline elements in the menu definitions, they will be overridden by this command.

The third example above specifies that autoslicing of elements will occur, and that every thick element will be cut into slices of approximately 0.01 meters. Note that this is only approximate; for example, an element with a length of 1.05 meter will be cut into 10 slices (i.e. the nearest integer equal to the element length divided by the slice length). Note also that when the element length is less than the sliced length, the number of slices is set equal to one.

2.2 Automatic application of a command or string of commands

Type Code: autoapply

Required Parameters:

1. *name:* name of a command in the menu, or name of a line that is a string of commands in the menu

Example:

```
myslice1: autoapply, name=myline
```

The example above specifies that, after every slice, the element or line with the name "myline" will be applied.

2.3 Automatic tracking

Type Code: autotrack

Required Parameters:

1. *type:* The type and order of tracking. Currently the following are valid: taylorN and symplecticN, where taylorN specifies that an Nth order taylor expansion will be used, and symplecticN specifies that a symplectic raytrace will be performed (based on an implicit procedure involving a generating function) and for which the symplectic raytrace will agree with the Taylor series approach through order N.

Example:

```
dotrack: autotrack, type=symplectic5
```

The example above specifies that, after every element or slice of an element (if autoslicing is enabled), a raytrace will be performed. The raytrace will be symplectic to machine precision, and it will agree with the Taylor series (or expanded Lie series) through 5th order.

2.4 Automatic concatenation

Type Code: autoconcat

Required Parameters: none

Example:

```
combinemaps: autoconcat
```

The example above specifies that, after every element or slice of an element (if autoslicing is enabled), the map that has just been computed will be concatenated with the total (accumulated) transfer map.

This is the default mode of operation for MaryLie and for MaryLie/IMPACT.

2.5 Set beam parameters

Type Code: beam

Parameters:

1. *particle:* proton, electron, positron, H-
2. *mass:* particle mass in GeV/c^2
3. *charge:* particle charge in units of e
4. *energy:* total energy in GeV
5. *ekinetic:* kinetic energy in GeV
6. *pc:* momentum in GeV/c
7. *brho:* magnetic rigidity in Tesla/meter
8. *gamma1:* $\gamma - 1$
9. *gamma:* γ
10. *bcurr:* beam current in Amperes.
11. *bfreq:* frequency of bunches in Hz.
12. *maxray:* maximum number of macroparticles to be created

This command is used to specify the parameters of a *reference particle*. This information is needed to compute maps (which are determined with respect to some reference trajectory). Also, a distribution of particles can be created that is specified in terms of deviations from the reference values.

Note that the the user is responsible for specifying the parameters sensibly. For example, the user can specify the particle and energy, or particle and momentum, or brho and gamma1, etc. The user should not specify the parameters in a way that is over- or under-determined.

Note that bfreq=f and bcurr=I determine the total charge per bunch, Q, through the relation $Q = I/f$. This is important when modeling beams with space charge.

Example:

```
setbeam: beam, particle=proton, ekinetic=0.800.
```

The example above specifies a proton beam with a kinetic energy of 0.800 GeV.

2.6 Print particle coordinates and momenta

Type Code: particledump

Parameters:

1. *min:*
2. *max:*
3. *sequencelength:*
4. *precision:*
5. *file:*
6. *close:*
7. *flush:*

Examples:

```
printrays: particledump, precision=5, file=outrays, sequencelength=100
print1to5: particledump, file=history.data, min=1, max=5, close=false
```

Whenever the first example is invoked, it will cause all the particle data to be printed on a new file called "outraysNNN", where NNN=001,002,003,... etc. The data will be printed with 5 digits of precision.

Whenever the second example is invoked, it will cause particles 1 through 5 to be printed on a file called "history.data" ;note that, if this command is automatically applied after every element or slice of an element, it will could be used to make ray plots of selected particles (in this case, the first 5 particles).

2.7 Set parameters of Poisson solver for space charge calculation

Type Code: poisson

Parameters:

1. *nx,ny,nz:* number of grid points in the x, y, and z directions
2. *boundingbox:* = *variable* (default) to specify that the code should automatically determine the grid size at every step so that the grid just enclosed the particles; = *fixed* to specify a fixed grid defined by xmin,xmax, ymin,ymax,zmin,zmax
3. *ngridpoints:* = *fixed* to keep nx,ny,nz fixed; = *variable* to specify that the code should adjust nx,ny,nz at each time step in such a way that the aspect ratio does not deviate significantly from 1.
4. *xmin,xmax:* = minimum and maximum grid values in x
5. *ymin,ymax:* = minimum and maximum grid values in y
6. *zmin,zmax:* = minimum and maximum grid values in z
7. *xboundary:* = *open, dirichlet, or periodic* boundary in the x-direction (default is OPEN, currently the only available option)
8. *yboundary:* = *open, dirichlet, or periodic* boundary in the y-direction (default is OPEN, currently the only available option)
9. *zboundary:* = *open, dirichlet, or periodic* boundary in the z-direction (default is OPEN, currently the only available option)

Example:

```
mypoisson: poisson, nx=32, ny=32, nz=32
```

The above specifies that the Poisson solver will use a 32^3 grid. Also, since everything else is based on the default values, the following will be in effect: the solver will assume open boundary conditions; the number of grid points will be fixed; and the code will automatically adjust the size of the bounding box (i.e. the box that surrounds the particles when open boundary conditions are in use), making the box bigger or smaller as the beam evolves so the the box is large enough to contain all the particles.

2.8 Print 1D Beam Profile

Type Code: profile1d

Parameters:

1. *column:* column number of 1D profile
2. *bins:* number of bins
3. *rwall:* wall radius (meters)
4. *sequencelength:* maximum number of files to be printed in a sequence of files
5. *precision:* decimal precision of data printed in files
6. *file:* file name
7. *close:* true (false) to close (not close) the current file
8. *flush:* true (false) to flush (not flush) the current file

Example:

```
myname: xxxxxxx, yyyy=...
```

The example above specifies...

2.9 rayscale: scale particle data

Type Code: rayscale

Required Parameters:

1. *xmult, xdiv:* Scale factor by which to multiply or divide x-data
2. *pxmult, pxdiv:* Scale factor by which to multiply or divide px-data
3. *ymult, ydiv:* Scale factor by which to multiply or divide y-data
4. *pymult, pydiv:* Scale factor by which to multiply or divide py-data
5. *tmult, tdiv:* Scale factor by which to multiply or divide t-data
6. *ptmult, ptdiv:* Scale factor by which to multiply or divide pt-data

Example:

```
myname: convert_from_cm, xdiv=100., ydiv=100.
```

The example scales x-data and y-data by dividing the values by 100. This could be used, for example, if the data that were read in happened to be produced by another code that specified x-data and y-data in centimeters, and the user wanted to convert the data to meters.

2.10 raytrace: trace rays

Type Code: raytrace

Required Parameters:

1. *norder:* order of ray trace
2. *ntrace:* to perform ntrace ray traces
3. *nwrite:* to write the data every nrwite'th time
4. *sequencelength:* maximum number of files to be printed in a sequence of files
5. *precision:* decimal precision of data printed in files
6. *min, max:* to print the particles in the range (min, max)
7. *file1:* name of input file
8. *file2:* name of output file, or base name if a sequence of files is to be printed
9. *close:* true (false) to close (not close) the output file
10. *flush:* true (false) to flush (not flush) the output file

Example:

```
mytrace: raytrace, norder=...
```

The example above specifies...

2.11 resetarclength: reset arc length

Type Code: resetarclength

Required Parameters: none

Example:

```
arclength0: resetarclength
```

The example above specifies...

2.12 symbolparser: control treatment of undefined symbols in input file

Type Code: symbolparser

Required Parameters:

1. *defaultzero:* "true" to assign undefined symbols the value zero; "false" (the default) to halt execution if undefined symbols are encountered in the input file.

Example:

```
undefined_is_zero: symbolparser, defaultzero=true
```

The example above specifies...

Chapter 3

MaryLie/IMPACT Examples

3.1 Overview

The MaryLie manual has many examples showing the use of MaryLie. The following contains additional examples that exhibit new features that are found in MaryLie/IMPACT. This includes, for example, beamlines with rf cavities that accelerate the beam, auto-slicing beamline elements, and auto-application of commands. Nearly all of the examples include the effects of space charge.

3.2 FODO Channel with RF Cavities . . . . . ??
3.3 KV Beam in a FODO Channel . . . . . ??
3.4 Virtual Wire Scanners in a FODO Channel . . . . . ??
3.5 Anistropic 2D Beam in a Constant Focusing Channel . . . . . ??
3.6 Thermal 3D Equilibrium in a Constant Focusing Channel . . . . . ??
3.7 Bi-Thermal 3D Equilibrium in a Constatnt Focusing Channel . . . . . ??

(Example chapter of this manual is in preparation.)